Document Technique

Projet: création d'une page inscription/connexion

Chef de Projet: Quentin LAVAL
Responsable Technique: Bastien FONTAGNE
Développeur: Kevin LEMAIRE

CIEL2

I.Introduction

Ce document explique le fonctionnement et la structure de notre système d' inscription/connexion pour notre site web. Le site permet de s’inscrire, de se connecter et de se déconnecter.

Sommaire:

2.Architecture

2.1.langage de programmation utilisé et technologie utilisé

Frontend: html/css et Javascript

Backend: Node.js

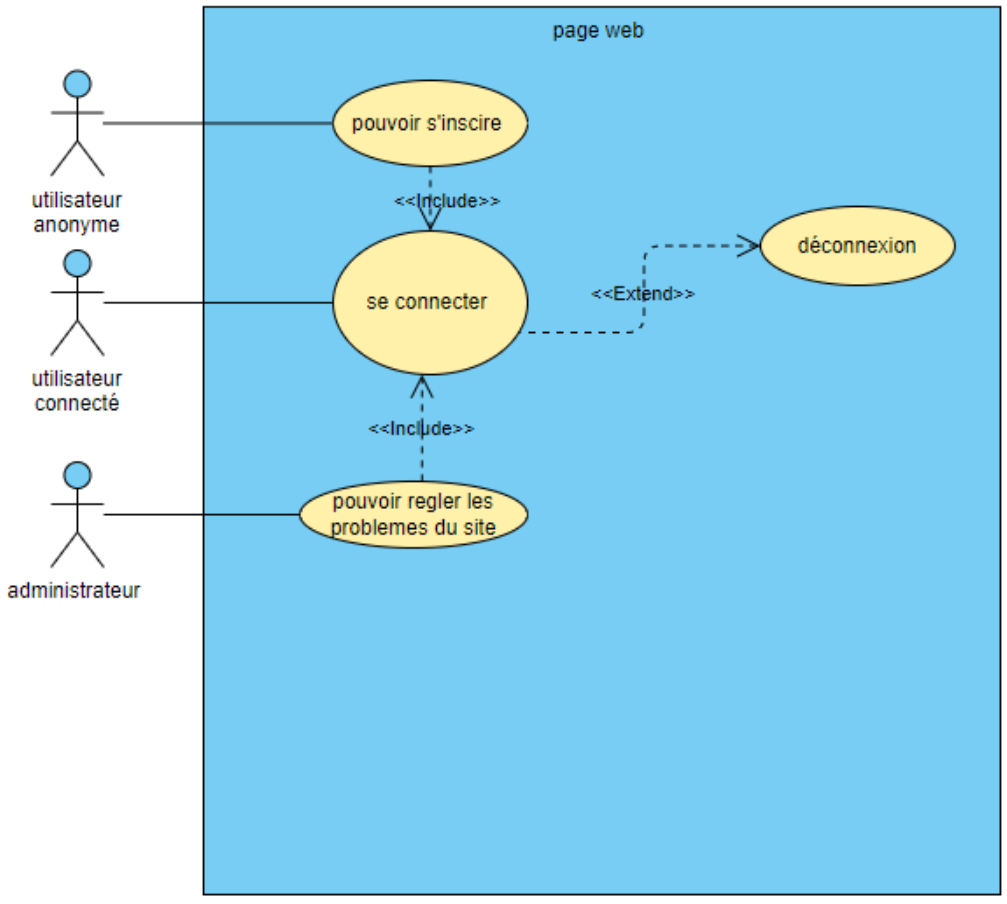
BDD: mysql/mariadb et phpmyadmin

host du site web: sur la vm en n°219 (ip: 192.168.64.162) avec apache

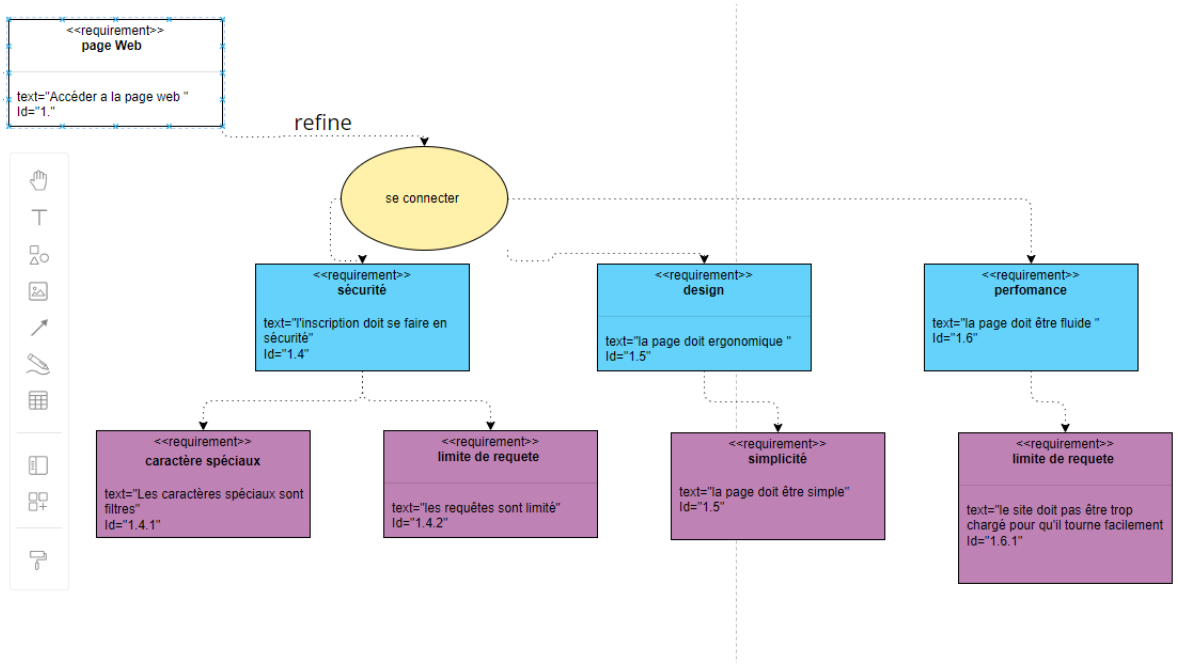
Authentication: Token

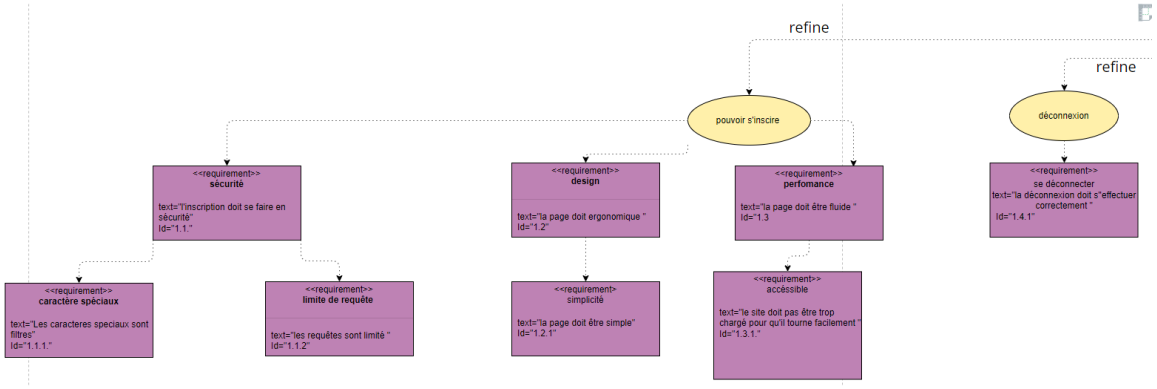
2.2.Diagramme de l’architecture du site web

2.3.Diagramme de use case



2.4.Diagramme d’exigence





2.5.Diagramme de Gantt prévisionnel

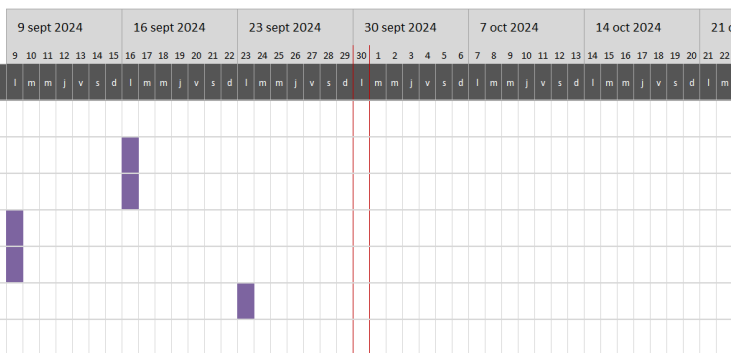
TITRE DU PROJET

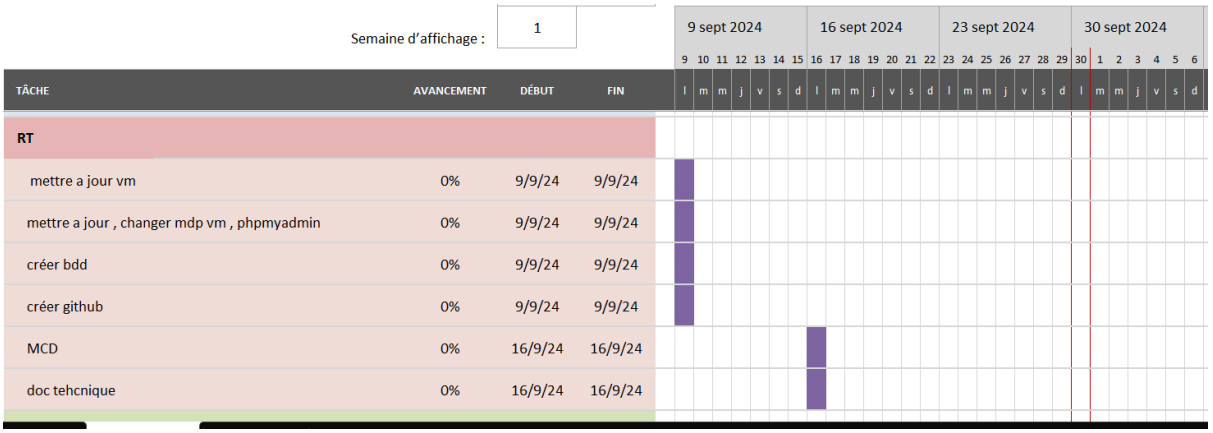
Nom de la société
Chef de projet

DIAGRAMME DE GANTT SIMPLE par Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

Début du projet : lun, 9/9/2024
Semaine d’affichage : 1

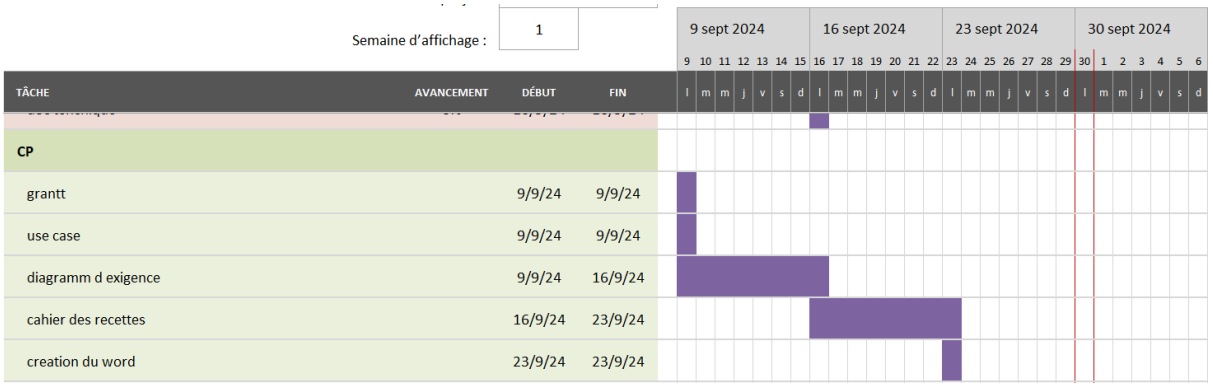
TÂCHE	AVANCEMENT	DÉBUT	FIN
dev			
page insciption	0%	16/9/24	16/9/24
fonctionnalité inscription	0%	16/9/24	16/9/24
page connexion	0%	9/9/24	9/9/24
fonctionalité connexion	0%	9/9/24	9/9/24
fonctionnalité deconnexion	0%	23/9/24	23/9/24





Semaine d'affichage : 1

TÂCHE	AVANCEMENT	DÉBUT	FIN
RT			
mettre a jour vm	0%	9/9/24	9/9/24
mettre a jour , changer mdp vm , phpmyadmin	0%	9/9/24	9/9/24
créer bdd	0%	9/9/24	9/9/24
créer github	0%	9/9/24	9/9/24
MCD	0%	16/9/24	16/9/24
doc tehcnique	0%	16/9/24	16/9/24



Semaine d'affichage : 1

TÂCHE	AVANCEMENT	DÉBUT	FIN
CP			
grantt		9/9/24	9/9/24
use case		9/9/24	9/9/24
diagramm d exigence		9/9/24	16/9/24
cahier des recettes		16/9/24	23/9/24
creation du word		23/9/24	23/9/24

3.Fonctionnalité

3.1.Inscription

Route: /register

Méthode: POST

Code: `//Route d'inscription via l'API`

```
app.post('/register', (req, res) => {

    const {mail, name, password} = req.body;
    if(!mail || !name || !password) return res.status(400).json({
message : "Le nom d'utilisateur ou le mot de passe est manquant !" });

    const sqlVerif = `SELECT mail FROM User WHERE mail = ?`

    connection.query(sqlVerif, [mail], (err, result) => {
        if(err) {
```

```
            console.error("Erreur lors de la vérification
d'inscription\nErreur SQL :\n",err);
            return res.status(500).json({ message: "Erreur lors de la
création du compte" });
        }

        if(result.length === 0) {
            bcrypt.genSalt(10, (err, salt) => {
                if(err) {
                    console.error("Erreur lors de la génération du sel
Bcrypt\nErreur Bcrpt Salt:\n",err);
                    return res.status(500).json({ message: "Erreur lors
de la création du compte" });
                }

                bcrypt.hash(password, salt, (hashErr, hashedPassword)
=> {
                    if (hashErr) {
                        console.error('Erreur lors du hachage du mot de
passe\nErreur Bcrypt Hash :\n', hashErr);
                        return res.status(500).json({ message: "Erreur
lors de la création du compte" });
                    }

                    let token = jwt.sign({ mail }, jwtKey);

                    const sqlNewUser = `INSERT INTO User (mail, name,
password, token) VALUES (?, ?, ?)`;

                    connection.query(sqlNewUser, [mail, hashedPassword,
token], (sqlErr) => {
                        if(sqlErr) {
                            console.error("Erreur lors de la création
du nouvel User\nErreur sql :\n", sqlErr);
                            return res.status(500).json({ message :
"Erreur lors de la création du compte" });
                        }
                        return res.status(200).json(
                            {
                                message: "Compte créé avec succès",
                                token: token
                            }
                        );
```

```
                    })
                })
            })
        } else {
            return res.status(409).json({ message : "Nom déjà utilisé
"})
        }
    })
});
```

Traitement:
-vérifie si le mail ou le nom ou le mot de passe sont manquant
-vérifie si il y a eu des erreurs lors de l'inscription
-vérifie si le mot de passe a était hacher et si il y a eu une erreur dans sa génération
-vérifie si il y a eu une erreur dans la création du compte
-vérifie si le nom a déja était utiliser
-créer le compte

Réponse:
200: "Compte créé avec avec succés"
400: "Le nom d'utilisateur ou le mot de passe sont manquants"
409: "Nom déjà utilisé"
500: "Erreur lors de la création du compte

3.2.Connexion

route: /login

méthode: POST

code: `//Route de connexion via l'API`

```
app.post('/login', (req, res) => {
    const { mail, password } = req.body;

    if(!mail || !password) return res.status(400).json({ message: "Nom
ou mot de passe manquant !" });

    const sqlLogin = 'SELECT password, token FROM User WHERE mail = ?'

    connection.query(sqlLogin, [mail], (err, result) => {
```

```
        if(err) {
            console.error("Erreur lors de la requête SQL login\nErreur
sql :\n",err);
            return res.status(500).json({ message: "Erreur lors de la
connexion" });
        }

        if(result.length === 0) {
            return res.status(409).json({ message: "Nom inconnu" });
        }

        const isPasswordValid = bcrypt.compareSync(password,
result[0].password);

        if(isPasswordValid) {
            const sqlLoginToken = 'UPDATE User SET token = ? mail = ?'
            let token = jwt.sign({ mail }, jwtKey);

            connection.query(sqlLoginToken, [token, mail], (err) => {
                if(err) {
                    console.error("Erreur lors de la requête
LoginToken\nErreur sql :\n", err);
                    return res.status(500).json({ message : " Erreur
lors de la création du token"});
                }

                return res.status(200).json(
                    {
                        message : "Connexion Réussie",
                        token : token
                    }
                )
            })
        } else {
            res.status(409).json({ message : "Mot de passe
incorrect"});
        }
    })
})
```

Traitement:
-vérifie si le mail, le nom ou le mot de passe sont manquant
-vérifie si il y a une erreur durant la requête sql
-vérifie si le nom est dans la bdd
-vérifie le token de connexion

Réponse:
200: “Connexion Réussie”
400: “Nom d’utilisateur ou mot de passe manquant”
409: “Nom inconnu” ou “Mot de passe incorrect”
500: “Erreur lors de la connexion” ou “Erreur lors de la création du token”

3.3.Déconnexion

route: /disconnect

méthode: GET

code:
```
//Route de déconnexion via l'API
app.post('/disconnect', (req, res) => {
    const {token} = req.body;

    if(!token) return res.status(400).json({ message: "Le token du User est requis pour la déconnexion"});

    const sqlDisconnect = 'UPDATE User SET token = NULL WHERE token = ?'

    connection.query(sqlDisconnect, [token], (err) => {
        if(err) {
            console.error("Erreur lors de la requête de déconnexion\n Erreur sql:\n", err);
            return res.status(500).json({ message : " Erreur lors de la déconnexion"});
        }
        return res.status(200).json({ message: " Utilisateur déconnecté"});
    })
})
```

Traitement:
-vérifie si il il y a le token de l'utilisateur requis pour la déconnexion
-vérifie la requête de déconnexion

Réponse:
200: "Utilisateur déconnecté"
400: "Le token du User est requis pour la déconnexion"
500: "Erreur lors de la déconnexion"

3.4Page Admin

route: /getUserList

méthode: GET

```
app.get('/getUserList', (req, res) => {
   const sqlUserList = 'SELECT mail, username FROM USER';

   connection.query(sqlUserList, (err, result) => {
      if (err) {
         console.error('Erreur lors de la requête sqlUserList\nErreur:\n', err);
         return res.status(500).json({ message: "Erreur lors de la récupération de la
liste des utilisateurs" });
      }

      let body = `
      <div class="navbar">
         <div class="sign-out-wrapper clearfix">
            <a href="#" id='logoutButton' class="sign-out pull-right">
               <span>Sign Out</span>
               <i class="fa fa-sign-out"></i>
            </a>
         </div>
      </div>
      <div class="content">
         <h1>Liste des utilisateurs</h1>
         <table border="1">
            <tr>
               <th>Email</th>
               <th>Nom d'utilisateur</th>
```

```
                <th>Actions</th>
            </tr>
        `;

        // Add each user to the table
        result.forEach(user => {
            body += `
                <tr>
                    <td>${user.mail}</td>
                    <td>${user.username}</td>
                    <td>
                        <button onclick="modifyPassword('${user.mail}')">Modifier le mot de
passe</button>
                        <button onclick="deleteAccount('${user.mail}')">Supprimer le
compte</button>
                    </td>
                </tr>
            `;
        });

        // Close the table and body tags
        body += `
                </table>
            </div>
            <div class="content" id="error">
            </div>
        `;

        return res.status(200).json({ body: body });
    });
});
```

route : /modifyPassword

méthode: POST

```
app.post('/modifyPassword', (req, res) => {
    const { email, newPassword } = req.body;
    const sqlUpdatePassword = 'UPDATE USER SET password = ? WHERE mail =
?';

    connection.query(sqlUpdatePassword, [newPassword, email], (err, result) => {
        if (err) {
            console.error('Erreur lors de la mise à jour du mot de passe\nErreur:\n', err);
```

```
            return res.status(500).json({ message: "Erreur lors de la mise à jour du mot
de passe" });
        }
        return res.status(200).json({ message: "Mot de passe mis à jour avec succès"
});
    });
});
```

route : /deleteAccount

méthode: GET

```
app.post('/deleteAccount', (req, res) => {
    const { email } = req.body;
    const sqlDeleteAccount = 'DELETE FROM USER WHERE mail = ?';

    connection.query(sqlDeleteAccount, [email], (err, result) => {
        if (err) {
            console.error('Erreur lors de la suppression du compte\nErreur:\n', err);
            return res.status(500).json({ message: "Erreur lors de la suppression du
compte" });
        }
        return res.status(200).json({ message: "Compte supprimé avec succès" });
    });
});

const authenticate = (req, res, next) => {
    const token = req.cookies.userToken;

    if (!token) {
        return res.redirect('/');
    }

    jwt.verify(token, jwtKey, (err, decoded) => {
        if (err) {
            return res.redirect('/');
        }

        req.user = decoded;
        next();
    });
};
```

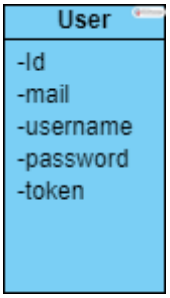
4.Sécurité

cryptage du mot de passe: les mots de passe sont hachés en utilisant bcrypte

token: les token sont créer en utilisant JWT, ils ont une durée limité et ont une clé secrète

5.Base De Donnée

5.1.Représentation visuel



6.Tests

6.1.Tests de l'inscription/connexion/déconnexion

Description	Critère d'acceptation	Résultat
1.Créer un compte	Le compte est créer avec un nom d'utilisateur, un e-mail et un mot de passe	ok
2.Se connecter au compte	On se connecte au compte en utilisant le nom d'utilisateur, le mail et le mot de passe associé	ok
3.Se déconnecter du compte	On se déconnecte du compte	ok

6.2.Test de la bdd

Description	Critère d'acceptation	Résultat
1.Création et stockage des donnès	On vérifie si les information de connexion sont bien stocker et crypter dans la bdd	ok
2.Création du token	On vérifie si le compte est bien liée à un token	ok

7.Déploiement

On déploie le site sur la vm n°219 (192.168.64.162)fr pour tester l'application

8.Conclusion

Ce document technique montre comment implémenter et sécuriser le système d'inscription et de connexion pour le site web, ce dernier sera enrichi au fur et à mesure des projets.

9.Annexe

9.1Mail:

1 sur 2

compte rendu le 16/09/2024 Boîte de réception ×

quentin laval <laval.quentin80@gmail.com> lun. 16 sept. 17:27
À julienlanglace, lemaire.kevin6020, fontagne.bastien1

Bonjours a tous voici le compte rendu de fin de journée

Nous avions prévu pour cette séance les choses suivantes
Kevin : pages de connections fonctionnelles
Bastien : faire le mcd et commencer le cahier de recette
Quentin faire le use case et commencer le diagramme d'exigence
Pour ma part je suis a l heure

Pour la semaine prochaine sa sera le dernier cours pour finaliser le projet
Kevin: finaliser le site
Quentin et Bastien : faire le diapo

Bastien Fontagne lun. 16 sept. 17:29
À moi

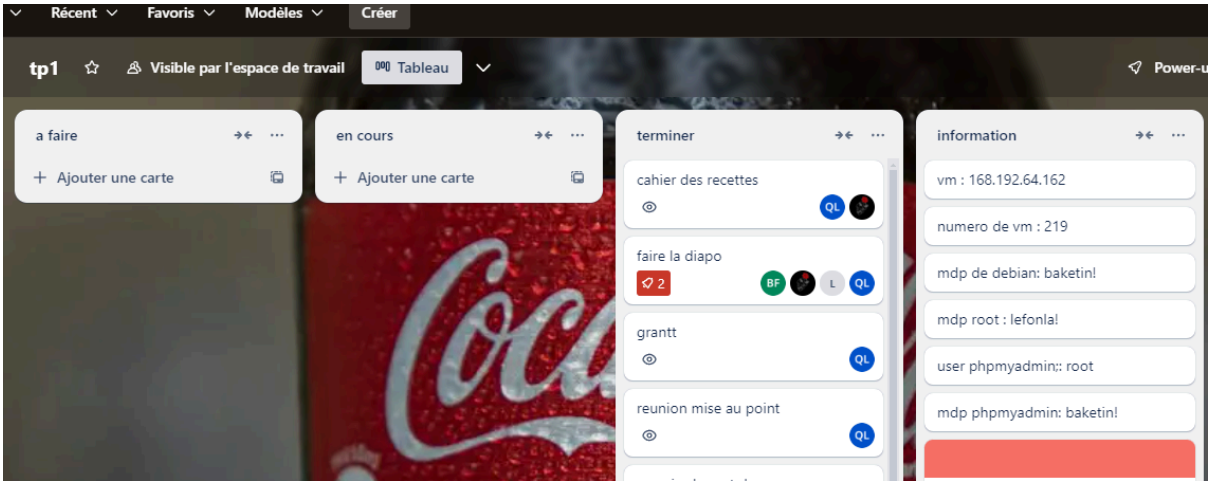
Tout est ok pour moi

Kevin Lemaire lun. 16 sept. 17:32
À moi

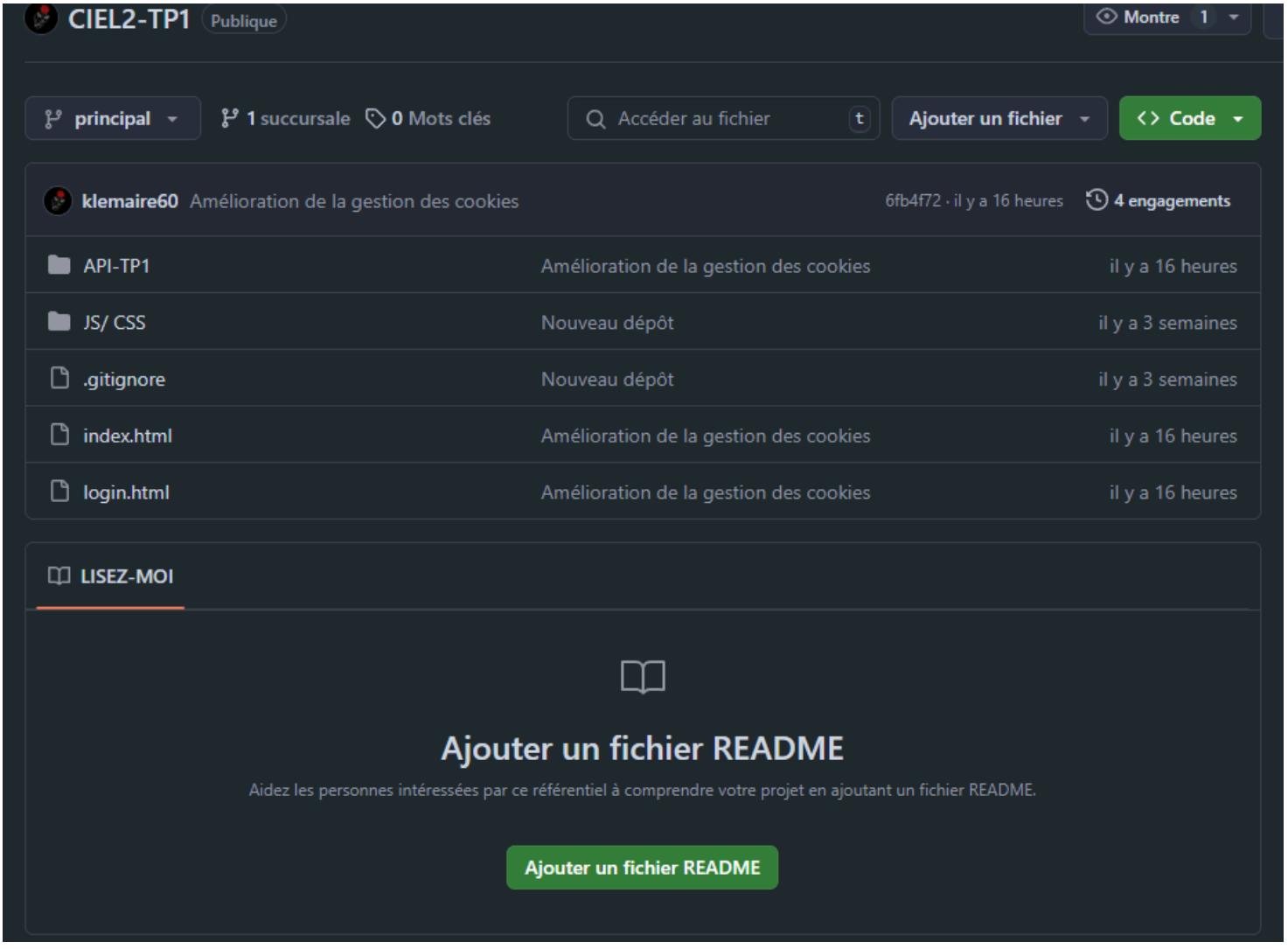
tout est bon pour moi

Répondre Transférer IA Réponse

9.2Trello:



9.3Github:


CIEL2-TP1
Publique
Montre 1
principal
1 succursale
0 Mots clés
Accéder au fichier
t
Ajouter un fichier
Code
klemaire60 Amélioration de la gestion des cookies
6fb4f72 · il y a 16 heures
4 engagements
API-TP1
Amélioration de la gestion des cookies
il y a 16 heures
JS/ CSS
Nouveau dépôt
il y a 3 semaines
.gitignore
Nouveau dépôt
il y a 3 semaines
index.html
Amélioration de la gestion des cookies
il y a 16 heures
login.html
Amélioration de la gestion des cookies
il y a 16 heures
LISEZ-MOI
Ajouter un fichier README
Aidez les personnes intéressées par ce référentiel à comprendre votre projet en ajoutant un fichier README.
Ajouter un fichier README